EXHIBIT "C"

```
#ifndef _IPLUGIN_IDL
#define _IPLUGIN_IDL

import "oaidl.idl";

// IPlugin

[
        object,
        uuid(980B1059-1AEB-11d3-B178-006094198F61),
        helpstring("IPlugin Interface"),
        pointer_default(unique)
]
interface IPlugin : IUnknown {
        [helpstring("method SetParent, sets an interface pointer to call back the plugin'
s parent to enable callback")]
        HRESULT SetParent([in] REFIID ParentIID, [in, iid_is(ParentIID)] void *pParent);
};

#endif // _IPLUGIN_IDL
```

DefaultDeliveryManager.cpp 1

```
// DefaultDeliveryManager.cpp : Implementation of CDefaultDeliveryManager
#include "stdafx.h"
#include "DefaultDeliveryManagerImpl.h"
#include "DefaultDeliveryManager.h"

HRESULT CDefaultDeliveryManager::FinalConstruct() {
        IDeliveryManagerImpl::FinalConstruct();
        m_pEnumChatLocation = NULL;
        return S_OK;
}

HRESULT CDefaultDeliveryManager::FinalRelease() {
        IDeliveryManagerImpl::FinalRelease();
        SAFERELEASE(m_pEnumChatLocation);
        return S_OK;
}

// IDeliveryManager

STDMETHODIMP CDefaultDeliveryManager::SetActiveChatLocations(REFIID EnumLocationsIID, voi
d *pEnumLocations) {
        if (!GUIDsAreEqual(EnumLocationsIID, __uuidof(IEnumChatLocation))) {
                return E_INVALIDARG;
        }
        m_pEnumChatLocation = static_cast<IEnumChatLocation *>(pEnumLocations);
        m_pEnumChatLocation->AddRef();
        m_pEnumChatLocation->AddRef();
        return S_OK;
}

STDMETHODIMP CDefaultDeliveryManager::Reset() {
        m_pEnumChatLocation->Reset();
        return S_OK;
}

STDMETHODIMP CDefaultDeliveryManager::StartDelivery(REFIID MessageIID, void *pMessage) {
        return TryNextLocation(MessageIID, pMessage);
}

STDMETHODIMP CDefaultDeliveryManager::MessageRouted(REFIID MessageIID, void *pMessage, RE
FIID LocationIID, void *pLocation, BOOL Succeeded, BOOL *pDone) {
        if (Succeeded) {
                *pDone = true;
                return S_OK;
        }
        return TryNextLocation(MessageIID, pMessage);
}

// Need to check HRESULTs here.
HRESULT CDefaultDeliveryManager::TryNextLocation(REFIID MessageIID, void *pMessage) {
        IChatLocation* pLocation = NULL;
        while (m_pEnumChatLocation->Next(1, &pLocation, NULL) == S_OK) {
                IChatUser *pUser;
                IChatMessage *pChatMessage = static_cast<IChatMessage *>(pMessage);
                pChatMessage->GetTo(__uuidof(IChatUser), (void **)&pUser);
                enum ChatUserStatus Status;
                HRESULT hr = pLocation->GetUserStatus(__uuidof(IChatUser), pUser, &Status
);
                if (FAILED(hr)) {
                        return E_FAIL;
                }
                if (hr == S_FALSE) {
                        continue;
                }
                if ((Status == CUS_ONLINE) || (Status == CUS_UNKNOWN)) {
                        hr = m_pDeliveryManagerWrapper->RouteMessage(MessageIID, pMessage
, __uuidof(IChatLocation), pLocation);
                        if (FAILED(hr)) {
```


```
                        return E_FAIL;
                    }
                    return S_OK;
                }
            }
            // All chat clients were unsuccessful.
            ::MessageBox(NULL, _T("Could not deliver message."), _T("Message Delivery Error")
, MB_OK);
            return S_OK;
}
```

ExtremeChat.cpp 1

```
// ExtremeChat.cpp : Implementation of CExtremeChat

#include "stdafx.h"

#define COM_MACROS_H_ACTUAL_IMPLEMENTATION
#include <com_macros.h>

#define IMPLEMENT_CONTINUE_NEW_THREAD

#include "ExtremeChatServer.h"
#include "ExtremeChat.h"

// Error and warning reporting

void CExtremeChat::ReportError(LPCTSTR msg) {
        WithScopeLocked(m_ReportErrorCritSec);

        // Need to concatentate with "Do you wish to debug?"
        int choice = ::MessageBox(NULL, msg, _T("ExtremeChat Error"), MB_YESNO);
        if (choice == IDYES) {
                DebugBreak();
        }
}

void CExtremeChat::ReportWarning(LPCTSTR msg) {
        WithScopeLocked(m_ReportErrorCritSec);

        // Need to concatentate with "Do you wish to debug?"
        int choice = ::MessageBox(NULL, msg, _T("ExtremeChat Warning"), MB_YESNO);
        if (choice == IDYES) {
                DebugBreak();
        }
}

// CExtremeChat

HRESULT CExtremeChat::FinalConstruct() {
        m_Initialized = false;
        m_NextMessageID = 0;
        m_pChatRegistry = NULL;
        m_pChatConfigMan = NULL;
        m_pExchatUI = NULL;
        m_pDeliveryManagerPool = NULL;
        return S_OK;
}

HRESULT CExtremeChat::FinalRelease() {
        SAFERELEASE(m_pChatRegistry);
        SAFERELEASE(m_pChatConfigMan);
        SAFERELEASE(m_pExchatUI);
        if (m_pDeliveryManagerPool != NULL) {
                delete m_pDeliveryManagerPool;
        }
        ReleaseMessagingClients();
        ReleaseChatClients();
        ReleaseMessageMap();
        ReleaseMessageIDMap();
        return S_OK;
}

// Chat client management

HRESULT CExtremeChat::CreateChatClient(IChatLocation *pLocation, IChatClientWrapper **ppW
rapper) {
        IID ChatClientIID;
        HRESULT hr = pLocation->GetPluginGUID(ChatClientIID);
        if (FAILED(hr)) {
                ReportError(_T("Could not get chat client GUID for location."));
```

```
                return E_FAIL;
        }

        IChatClientWrapper *pChatClientWrapper = NULL;
        hr = CoCreateInstance(EC_CLSID_DefaultChatClientWrapper, NULL, CLSCTX_ALL, __uuid
of(IChatClientWrapper), (void **)&pChatClientWrapper);
        if (FAILED(hr)) {
                ReportError(_T("Failed to create chat client wrapper."));
                return E_FAIL;
        }

        hr = pChatClientWrapper->SetParent(__uuidof(IChatClientListener), static_cast<ICh
atClientListener *>(this));
        if (FAILED(hr)) {
                ReportError(_T("Could not set chat client wrapper parent."));
                return E_FAIL;
        }

        IChatClient *pChatClient = NULL;
        hr = CoCreateInstance(ChatClientIID, NULL, CLSCTX_ALL, __uuidof(IChatClient), (vo
id **)&pChatClient);
        if (FAILED(hr)) {
                ReportError(_T("Failed to create chat client."));
                return E_FAIL;
        }

        hr = pChatClientWrapper->SetPlugin(__uuidof(IChatClient), (void **)pChatClient);
        if (FAILED(hr)) {
                ReportError(_T("Could not set chat client wrapper plugin."));
                return E_FAIL;
        }
        pChatClient->Release();

        hr = pChatClientWrapper->SetChatConfigMan(__uuidof(IChatConfigMan), m_pChatConfig
Man);
        if (FAILED(hr)) {
                ReportError(_T("Failed to set configuration manager for new chat client w
rapper."));
                return E_FAIL;
        }

        hr = pChatClientWrapper->UpdateLocation(__uuidof(IChatLocation), pLocation);
        if (FAILED(hr)) {
                ReportError(_T("Failed to update location for new chat client wrapper."))
;
                return E_FAIL;
        }

        *ppWrapper = pChatClientWrapper;
        return S_OK;
}

HRESULT CExtremeChat::ReleaseChatClients() {
        for (LOCATION_CHATCLIENTWRAP_MAP::iterator iter = m_LocationMap.begin(); iter !=
m_LocationMap.end(); iter++) {
                IChatClientWrapper *pChatClientWrapper = iter->second;
                pChatClientWrapper->Release();
        }
        m_LocationMap.clear();
        return S_OK;
}

HRESULT CExtremeChat::ConfigureChatClients() {
        HRESULT hr = ReleaseChatClients();
        if (FAILED(hr)) {
                ReportError(_T("Could not release active chat clients."));
                return E_FAIL;
        }
```

```
        // Retreive an enumeration of all known locations.
        IEnumChatLocation *pEnumLocations = NULL;
        hr = m_pChatConfigMan->GetEnumLocations(__uuidof(IEnumChatLocation), (void**)&pEn
umLocations);
        if (FAILED(hr)) {
                ReportError(_T("Could not allocate location enumerator."));
                return E_FAIL;
        }
        hr = pEnumLocations->Reset();
        if (FAILED(hr)) {
                ReportError(_T("Failed to reset location enumerator."));
                return E_FAIL;
        }

        // Create plugins that are selected in the chat registry.
        IChatLocation* pLocation = NULL;
        while (pEnumLocations->Next(1, &pLocation, NULL) == S_OK) {
                IID LocationCLSID;
                hr = pLocation->GetPluginGUID(LocationCLSID);
                if (FAILED(hr)) {
                        ReportError(_T("Could not get plugin CLSID for location."));
                        return E_FAIL;
                }

                ChatRegistryInfo Info;
                hr = m_pChatRegistry->RegistryLookupPlugin(LocationCLSID, ECRIT_ALL, &Inf
o);
                if (FAILED(hr)) {
                        ReportError(_T("Could not query chat registry for chat plugin sel
ection."));
                        return E_FAIL;
                }
                if (hr == S_FALSE) {
                        ReportWarning(_T("Chat client for location not found in chat regi
stry."));
                        continue;
                }

                if (!Info.selected) {
                        // ReportWarning(_T("Chat client for location not selected in cha
t registry."));
                        continue;
                }

                IChatClientWrapper *pWrapper = NULL;
                HRESULT hr = CreateChatClient(pLocation, &pWrapper);
                if (FAILED(hr)) {
                        ReportError(_T("Could not create chat client for location."));
                        return E_FAIL;
                }

                m_LocationMap[pLocation] = pWrapper;
        }
        return S_OK;
}

// Messaging client management

HRESULT CExtremeChat::CreateMessagingClient(ChatRegistryInfo *pInfo, IMessagingClientWrap
per **ppWrapper) {
        IMessagingClientWrapper *pMessagingClientWrapper = NULL;
        HRESULT hr = CoCreateInstance(EC_CLSID_DefaultMessagingClientWrapperObj, NULL, CL
SCTX_ALL, __uuidof(IMessagingClientWrapper), (void**)&pMessagingClientWrapper);
        if (FAILED(hr)) {
                ReportError(_T("Failed to create messaging client wrapper."));
                return E_FAIL;
        }
```

ExtremeChat.cpp 4

```
        hr = pMessagingClientWrapper->SetParent(__uuidof(IMessagingClientListener), stati
c_cast<IMessagingClientListener*>(this));
        if (FAILED(hr)) {
                ReportError(_T("Failed to set parent for messaging client wrapper."));
                return E_FAIL;
        }

        // Create messaging client according to its registry type.
        IMessagingClient* pMessagingClient = NULL;
        switch (pInfo->types) {

        case ECPT_UI:
                IID MessagingClientIID;
                hr = m_pExchatUI->CreateUserInterfacePlugin(pInfo->clsid, MessagingClient
IID, (void **)&pMessagingClient);
                if (FAILED(hr)) {
                        ReportError(_T("Could not create UI messaging client."));
                        return E_FAIL;
                }
                if (!GUIDsAreEqual(MessagingClientIID, __uuidof(IMessagingClient))) {
                        ReportError(_T("Illegal IID for new UI messaging client."));
                        return E_FAIL;
                }
                break;

        case ECPT_MESSAGING:
                hr = CoCreateInstance(pInfo->clsid, NULL, CLSCTX_ALL, __uuidof(IMessaging
Client), (void**)&pMessagingClient);
                if (FAILED(hr)) {
                        ReportError(_T("Could not create messaging client."));
                        return E_FAIL;
                }
                break;

        default:
                ReportError(_T("Unsupported plugin type for messaging client."));
                return E_FAIL;
        }

        // Set configuration manager
        hr = pMessagingClientWrapper->UpdateConfigurationManager(__uuidof(IChatConfigMan)
, (IUnknown *) m_pChatConfigMan);
        if (FAILED(hr)) {
                ReportError(_T("Failed to update configuration manager."));
                return E_FAIL;
        }

        // Set wrapper plugin.
        hr = pMessagingClientWrapper->SetPlugin(__uuidof(IMessagingClient), (IUnknown*)pM
essagingClient);
        if (FAILED(hr)) {
                ReportError(_T("Failed to set plugin for UI messaging client wrapper."));
                return E_FAIL;
        }
        pMessagingClient->Release();

        // Get local user.
        IID LocalUserIID;
        IChatUser *pLocalUser = NULL;
        hr = m_pChatConfigMan->GetLocalUser(LocalUserIID, (void **)&pLocalUser);
        if (FAILED(hr)) {
                ReportError(_T("Failed to get local user."));
                return E_FAIL;
        }
        if (!GUIDsAreEqual(LocalUserIID, __uuidof(IChatUser))) {
                ReportError(_T("Illegal IID for local user."));
                return E_FAIL;
```

```
        }

        // Set local user.
        hr = pMessagingClientWrapper->UpdateLocalUser(__uuidof(IChatUser), pLocalUser);
        if (FAILED(hr)) {
                ReportError(_T("Failed to update local user."));
                return E_FAIL;
        }

        hr = pMessagingClientWrapper->UpdateExchatUI(__uuidof(IExchatUI), (IUnknown*)m_pE
xchatUI);
        if (FAILED(hr)) {
                ReportError(_T("Failed to update ExchatUI for messaging client wrapper.")
);
                return E_FAIL;
        }

        // Get users list
        IEnumChatUser* pEnumChatUser;
        hr = m_pChatConfigMan->GetEnumUsers(__uuidof(IEnumChatUser), (void**) &pEnumChatU
ser);
        if (FAILED(hr)) {
                ReportError(_T("Failed get users list."));
                return E_FAIL;
        }

        // Set users list
        hr = pMessagingClientWrapper->UpdateUsersList(__uuidof(IEnumChatUser), (IUnknown
*)pEnumChatUser);
        if (FAILED(hr)) {
                ReportError(_T("Failed to update users list."));
                return E_FAIL;
        }

        // Get groups list
        IEnumChatGroup* pEnumChatGroup;
        hr = m_pChatConfigMan->GetEnumGroups(__uuidof(IEnumChatGroup), (void**) &pEnumCha
tGroup);
        if (FAILED(hr)) {
                ReportError(_T("Failed get groups list."));
                return E_FAIL;
        }

        // Set groups list
        hr = pMessagingClientWrapper->UpdateGroupsList(__uuidof(IEnumChatGroup), (IUnknow
n *)pEnumChatGroup);
        if (FAILED(hr)) {
                ReportError(_T("Failed to update groups list."));
                return E_FAIL;
        }

        *ppWrapper = pMessagingClientWrapper;
        return S_OK;
}

HRESULT CExtremeChat::ReleaseMessagingClients() {
        for (MESSAGINGCLIENTWRAP_SET::iterator iter = m_MessagingClients.begin(); iter !=
 m_MessagingClients.end(); iter++) {
                (*iter)->Release();
        }
        m_MessagingClients.clear();
        return S_OK;
}

HRESULT CExtremeChat::ConfigureMessagingClients() {
        HRESULT hr = ReleaseMessagingClients();
        if (FAILED(hr)) {
                ReportError(_T("Could not release active messaging clients."));
```

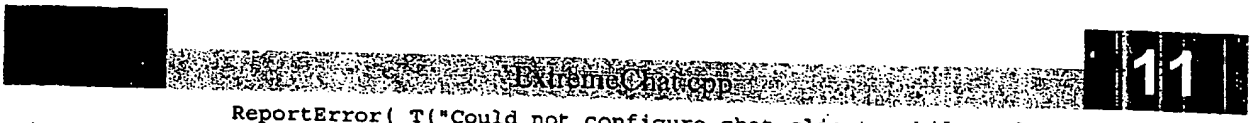
```
                return E_FAIL;
        }

        // Search chat registry for all types of messaging clients.
        short nMatches = 0;
        ChatRegistryInfo* pMatches = NULL;
        hr = m_pChatRegistry->EnumPlugins(ECPT_UI | ECPT_MESSAGING, ECRIT_CLSID | ECRIT_T
YPES | ECRIT_SELECTED, &nMatches, &pMatches);
        if (FAILED(hr)) {
                ReportError(_T("Could not query chat registry for messaging clients."));
                return E_FAIL;
        }

        BOOL bFoundActiveUI = false;
        for (int idx = 0; idx < nMatches; idx++) {
                ChatRegistryInfo *pInfo = &pMatches[idx];

                if (!pInfo->selected) {
                        continue;
                }

                if (pInfo->types == ECPT_UI) {
                        if (bFoundActiveUI) {
                                ReportError(_T("Multiple active UI messaging clients foun
d in chat registry."));
                                return E_FAIL;
                        }
                        bFoundActiveUI = true;
                }

                IMessagingClientWrapper *pMessagingClientWrapper = NULL;
                hr = CreateMessagingClient(pInfo, &pMessagingClientWrapper);
                if (FAILED(hr)) {
                        ReportError(_T("Could not create messaging client."));
                        return E_FAIL;
                }

                m_MessagingClients.insert(pMessagingClientWrapper);
        }

        return S_OK;
}

// Delivery manager pool

HRESULT CExtremeChat::CreateEnumChatLocation(IEnumChatLocation **ppEnumChatLocation) {
        // Create object.
        CComObject<CoEnumListLocation>* pEnum = NULL;
        HRESULT hr = CComObject<CoEnumListLocation>::CreateInstance(&pEnum);
        if (FAILED(hr)) {
                ReportError(_T("Failed to create object for location enumerator."));
                return E_FAIL;
        }
        pEnum->AddRef();

        // Construct location list.
        LOCATION_LIST LocationList;
        for (LOCATION_CHATCLIENTWRAP_MAP::iterator iter = m_LocationMap.begin(); iter !=
m_LocationMap.end(); iter++) {
                LocationList.push_back(iter->first);
        }

        // Create container copy.
        CComObject< CComContainerCopy< list<IChatLocation *>, CComMultiThreadModel > > *p
Copy = NULL;
        hr = pCopy->CreateInstance(&pCopy);
        if (FAILED(hr)) {
                ReportError(_T("Failed to create container copy for location enumerator."
```

```
));
                return E_FAIL;
        }
        pCopy->AddRef();

        // Initialize copy.
        hr = pCopy->Copy(LocationList);
        if (FAILED(hr)) {
                ReportError(_T("Failed to copy list for location enumerator."));
                return E_FAIL;
        }

        // Init enumerator with copy
        hr = pEnum->Init(pCopy->GetUnknown(), pCopy->m_coll);
        if (FAILED(hr)) {
                ReportError(_T("Failed to initialize location enumerator with container c
opy."));
                return E_FAIL;
        }

        SAFERELEASE(pCopy);
        *ppEnumChatLocation = pEnum;
        return S_OK;
}

HRESULT CExtremeChat::ConfigureDeliveryManagerPool() {
        if (m_pDeliveryManagerPool != NULL) {
                delete m_pDeliveryManagerPool;
                m_pDeliveryManagerPool = NULL;
        }
        m_pDeliveryManagerPool = new DeliveryManagerPool(static_cast<IDeliveryManagerList
ener *>(this));

        // Build an enumeration of all known locations.
        IEnumChatLocation *pEnumLocations = NULL;
        HRESULT hr = CreateEnumChatLocation(&pEnumLocations);
        if (FAILED(hr)) {
                ReportError(_T("Could not create location enumerator."));
                return E_FAIL;
        }

        m_pDeliveryManagerPool->SetActiveChatLocations(pEnumLocations);
        SAFERELEASE(pEnumLocations);
        return S_OK;
}

// Message map management

HRESULT CExtremeChat::MessageMapInsert(IChatMessage* pChatMessage, IMessagingClientWrappe
r *pMessagingClientWrapper) {
        pChatMessage->AddRef();
        pMessagingClientWrapper->AddRef();
        m_MessageMap[pChatMessage] = pMessagingClientWrapper;
        return S_OK;
}

HRESULT CExtremeChat::MessageMapRemove(IChatMessage* pChatMessage) {
        MESSAGE_MESSAGINGCLIENTWRAP_MAP::iterator iter = m_MessageMap.find(pChatMessage);
        if (iter == m_MessageMap.end()) {
                ReportError(_T("Could not find message for removal from map."));
                return E_FAIL;
        }

        pChatMessage->Release();
        // Release messaging client
        iter->second->Release();
        m_MessageMap.erase(iter);
```

```
        return S_OK;
}

HRESULT CExtremeChat::ReleaseMessageMap() {
        for (MESSAGE_MESSAGINGCLIENTWRAP_MAP::iterator iter = m_MessageMap.begin(); iter
!= m_MessageMap.end(); iter++) {
                HRESULT hr = MessageMapRemove(iter->first);
                if (FAILED(hr)) {
                        ReportError(_T("Failed to remove message from map."));
                        return E_FAIL;
                }
        }
        return S_OK;
}

// Message ID map management

HRESULT CExtremeChat::MessageIDMapInsert(IChatMessage* pChatMessage, IDeliveryManagerWrap
per* pDeliveryManagerWrapper, IChatLocation* pLocation) {
        MessageInfo *pMessageInfo = new MessageInfo;
        if (pMessageInfo == NULL) {
                ReportError(_T("Failed to create message info."));
                return E_FAIL;
        }

        pMessageInfo->pChatMessage = pChatMessage;
        pChatMessage->AddRef();

        pMessageInfo->pDeliveryManagerWrapper = pDeliveryManagerWrapper;
        pDeliveryManagerWrapper->AddRef();

        pMessageInfo->pLocation = pLocation;
        pLocation->AddRef();

        HRESULT hr = pChatMessage->SetID(m_NextMessageID);
        if (FAILED(hr)) {
                ReportError(_T("Failed to set message ID."));
                return E_FAIL;
        }

        m_MessageIDMap[m_NextMessageID] = pMessageInfo;
        m_NextMessageID++;

        return S_OK;
}

HRESULT CExtremeChat::MessageIDMapRemove(long MessageID) {
        LONG_MESSAGEINFO_MAP::iterator iter = m_MessageIDMap.find(MessageID);
        if (iter == m_MessageIDMap.end()) {
                ReportError(_T("Could not find message ID for removal from map."));
                return E_FAIL;
        }

        MessageInfo *pMessageInfo = iter->second;
        pMessageInfo->pChatMessage->Release();
        pMessageInfo->pDeliveryManagerWrapper->Release();
        pMessageInfo->pLocation->Release();
        delete pMessageInfo;
        m_MessageIDMap.erase(iter);

        return S_OK;
}

HRESULT CExtremeChat::ReleaseMessageIDMap() {
        for (LONG_MESSAGEINFO_MAP::iterator iter = m_MessageIDMap.begin(); iter != m_Mess
ageIDMap.end(); iter++) {
                HRESULT hr = MessageIDMapRemove(iter->first);
                if (FAILED(hr)) {
```

```
                        ReportError(_T("Failed to remove message ID from map."));
                        return E_FAIL;
                }
        }
        return S_OK;
}

// IExtremeChat

STDMETHODIMP CExtremeChat::SetExchatUI(REFIID ExchatUIIID, IUnknown *pExchatUI) {
        WithScopeLocked(m_CritSec);

        CoerceInterfacePointer(ExchatUIIID, pExchatUI, IExchatUI, m_pExchatUI);

        HRESULT hr = CoCreateInstance(EC_CLSID_ChatRegistry, NULL, CLSCTX_ALL, __uuidof(I
ChatRegistry), (void**)&m_pChatRegistry);
        if (FAILED(hr)) {
                ReportError(_T("Failed to create chat registry."));
                return E_FAIL;
        }

        LPOLESTR sJupFile, sUserFile, sSkinFile;
        hr = m_pChatRegistry->GetConfigurationString(OLESTR("JuplistFile"), &sJupFile);
        if (FAILED(hr)) {
                ReportError(_T("Could not find juplist file in chat registry."));
                return E_FAIL;
        }
        hr = m_pChatRegistry->GetConfigurationString(OLESTR("UserFile"), &sUserFile);
        if (FAILED(hr)) {
                ReportError(_T("Could not find user file in chat registry."));
                return E_FAIL;
        }

        hr = m_pChatRegistry->GetConfigurationString(OLESTR("SkinFile"),&sSkinFile);
        if (FAILED(hr)) {
                ReportError(_T("Could not find skin file in chat registry."));
                return E_FAIL;
        }

        hr = CoCreateInstance(EC_CLSID_ChatConfigMan, NULL, CLSCTX_ALL, __uuidof(IChatCon
figMan), (void**)&m_pChatConfigMan);
        if (FAILED(hr)) {
                ReportError(_T("Failed to create configuration manager."));
                return E_FAIL;
        }

        hr = m_pChatConfigMan->LoadSettings(sJupFile, sUserFile);
        if (FAILED(hr)) {
                ReportError(_T("Could not load configuration manager settings."));
                return E_FAIL;
        }

        // This should not be exposed at this level.
        IChatSkinProvider *pSkinProv = NULL;
        hr = m_pChatConfigMan->QueryInterface(__uuidof(IChatSkinProvider),(void**)&pSkinP
rov);
        if (FAILED(hr)) {
                ReportError(_T("Unable to query configuration manager for IChatSkinProvid
er."));
                return E_FAIL;
        }
        hr = pSkinProv->LoadSkin(sSkinFile);
        if (FAILED(hr)) {
                ReportError(_bstr_t("LoadSkin call failed"));
        }
        pSkinProv->Release();

        hr = m_pExchatUI->SetConfigurationPointer(__uuidof(IChatConfigMan), m_pChatConfig
```

```
Man);
        if (FAILED(hr)) {
                ReportError(_T("Failed to set configuration manager for UI component."));
                return E_FAIL;
        }

        hr = ConfigureChatClients();
        if (FAILED(hr)) {
                ReportError(_T("Failed to configure chat clients."));
                return E_FAIL;
        }

        hr = ConfigureMessagingClients();
        if (FAILED(hr)) {
                ReportError(_T("Failed to configure messaging clients."));
                return E_FAIL;
        }

        hr = ConfigureDeliveryManagerPool();
        if (FAILED(hr)) {
                ReportError(_T("Failed to configure delivery manager pool."));
                return E_FAIL;
        }

        m_Initialized = true;
        return S_OK;
}

STDMETHODIMP CExtremeChat::PluginsUpdated() {
        // For now, we just rebuild everything.  We should be more graceful to avoid drop
ping outstanding
        // messages unnecessarily.

        WithScopeLocked(m_CritSec);

        if (!m_Initialized) {
                return E_UNEXPECTED;
        }

        HRESULT hr = ReleaseMessagingClients();
        if (FAILED(hr)) {
                ReportError(_T("Could not release messaging clients while updating plugin
s."));
                return E_FAIL;
        }

        hr = ReleaseChatClients();
        if (FAILED(hr)) {
                ReportError(_T("Could not release chat clients while updating plugins."))
;
                return E_FAIL;
        }

        hr = ReleaseMessageMap();
        if (FAILED(hr)) {
                ReportError(_T("Could not release message map while updating plugins."));
                return E_FAIL;
        }

        hr = ReleaseMessageIDMap();
        if (FAILED(hr)) {
                ReportError(_T("Could not release message ID map while updating plugins."
));
                return E_FAIL;
        }

        hr = ConfigureChatClients();
        if (FAILED(hr)) {
```




```
                ReportError(_T("Could not configure chat clients while updating plugins."
));
                return E_FAIL;
        }

        hr = ConfigureMessagingClients();
        if (FAILED(hr)) {
                ReportError(_T("Could not configure messaging clients while updating plug
ins."));
                return E_FAIL;
        }

        hr = ConfigureDeliveryManagerPool();
        if (FAILED(hr)) {
                ReportError(_T("Failed to configure delivery manager pool while updating
plugins."));
                return E_FAIL;
        }

        return S_OK;
}

// IMessagingClientListener

STDMETHODIMP CExtremeChat::SendMessage(REFIID ClientWrapperIID, IUnknown *pClientWrapper,
 REFIID MessageIID, IUnknown *pMessage) {
        IMessagingClientWrapper *pMessagingClientWrapper = NULL;
        CoerceInterfacePointer(ClientWrapperIID, pClientWrapper, IMessagingClientWrapper,
 pMessagingClientWrapper);

        IChatMessage *pChatMessage = NULL;
        CoerceInterfacePointer(MessageIID, pMessage, IChatMessage, pChatMessage);

        WithScopeLocked(m_CritSec);

        if (!m_Initialized) {
                return E_UNEXPECTED;
        }

        HRESULT hr = pChatMessage->SetStatus(CMS_INPROGRESS);
        if (FAILED(hr)) {
                ReportError(_T("Failed to set status of new message."));
                return E_FAIL;
        }

        hr = MessageMapInsert(pChatMessage, pMessagingClientWrapper);
        if (FAILED(hr)) {
                ReportError(_T("Failed to insert new message into message map."));
                return E_FAIL;
        }

        IDeliveryManagerWrapper *pDeliveryManagerWrapper = m_pDeliveryManagerPool->Alloca
teDeliveryManager();

        // Continue in new thread.
        pDeliveryManagerWrapper->AddRef();
        pChatMessage->AddRef();
        hr = ContinueNewThread(SendMessageContinue, 3, this, (void *)pDeliveryManagerWrap
per, (void *)pChatMessage);
        if (FAILED(hr)) {
                ReportError(_T("Failed to continue sending message in new thread."));
                return E_FAIL;
        }

        SAFERELEASE(pMessagingClientWrapper);
        SAFERELEASE(pChatMessage);
        return S_OK;
}
```

```
HRESULT SendMessageContinue(CExtremeChat *pExtremeChat, void *pvDeliveryManagerWrapper, v
oid *pvChatMessage) {
        IDeliveryManagerWrapper *pDeliveryManagerWrapper = (IDeliveryManagerWrapper *)pvD
eliveryManagerWrapper;
        IChatMessage *pChatMessage = (IChatMessage *)pvChatMessage;

        HRESULT hr = pDeliveryManagerWrapper->StartDelivery(__uuidof(IChatMessage), pChat
Message);
        if (FAILED(hr)) {
                pExtremeChat->ReportError(_T("Failed to start delivery manager."));
                return E_FAIL;
        }

        SAFERELEASE(pDeliveryManagerWrapper);
        SAFERELEASE(pChatMessage);
        return S_OK;
}

STDMETHODIMP CExtremeChat::SetLocalStatus(enum ChatUserStatus Status) {
        WithScopeLocked(m_CritSec);

        if (!m_Initialized) {
                return E_UNEXPECTED;
        }

        for (LOCATION_CHATCLIENTWRAP_MAP::iterator iter = m_LocationMap.begin(); iter !=
m_LocationMap.end(); iter++) {
                IChatClientWrapper *pChatClientWrapper = iter->second;

                // Continue in new thread.
                pChatClientWrapper->AddRef();
                HRESULT hr = ContinueNewThread(SetLocalStatusContinue, 3, this, (void *)p
ChatClientWrapper, (void *)Status);
                if (FAILED(hr)) {
                        ReportError(_T("Failed to continue setting local status in new th
read."));
                        return E_FAIL;
                }
        }

        return S_OK;
}

HRESULT SetLocalStatusContinue(CExtremeChat *pExtremeChat, void *pvChatClientWrapper, voi
d *pvStatus) {
        IChatClientWrapper *pChatClientWrapper = (IChatClientWrapper *)pvChatClientWrappe
r;
        enum ChatUserStatus Status = (enum ChatUserStatus)((long)pvStatus);

        HRESULT hr = pChatClientWrapper->SetUserStatus(Status);
        if (FAILED(hr)) {
                pExtremeChat->ReportError(_T("Failed to set user status for chat client."
));
                return E_FAIL;
        }

        SAFERELEASE(pChatClientWrapper);
        return S_OK;
}


// IDeliveryManagerListener

STDMETHODIMP CExtremeChat::RouteMessage(REFIID ManagerIID, void *pManager, REFIID Message
IID, void *pMessage, REFIID LocationIID, void *pLocation) {
        IDeliveryManagerWrapper *pDeliveryManagerWrapper = NULL;
        CoerceInterfacePointer(ManagerIID, pManager, IDeliveryManagerWrapper, pDeliveryMa
```

```
nagerWrapper);

        IChatMessage *pNewMessage = NULL;
        CoerceInterfacePointer(MessageIID, pMessage, IChatMessage, pNewMessage);

        IChatLocation *pChatLocation = NULL;
        CoerceInterfacePointer(LocationIID, pLocation, IChatLocation, pChatLocation);

        WithScopeLocked(m_CritSec);

        if (!m_Initialized) {
                return E_UNEXPECTED;
        }

        HRESULT hr = MessageIDMapInsert(pNewMessage, pDeliveryManagerWrapper, pChatLocati
on);
        if (FAILED(hr)) {
                ReportError(_T("Failed to add message ID to routing table."));
                return E_FAIL;
        }

        // Find chat client for routing location.
        LOCATION_CHATCLIENTWRAP_MAP::iterator iter = m_LocationMap.find(pChatLocation);
        if (iter == m_LocationMap.end()) {
                ReportError(_T("Location not found while routing message."));
                return E_FAIL;
        }
        IChatClientWrapper *pClientWrapper = iter->second;

        // Continue in new thread.
        pNewMessage->AddRef();
        pClientWrapper->AddRef();
        hr = ContinueNewThread(RouteMessageContinue, 3, this, (void *)pNewMessage, (void
*)pClientWrapper);
        if (FAILED(hr)) {
                ReportError(_T("Failed to continue routing message in new thread."));
                return E_FAIL;
        }

        SAFERELEASE(pDeliveryManagerWrapper);
        SAFERELEASE(pNewMessage);
        SAFERELEASE(pChatLocation);
        return S_OK;
}

HRESULT RouteMessageContinue(CExtremeChat *pExtremeChat, void *pvChatMessage, void *pvCha
tClientWrapper) {
        IChatMessage *pMessage = (IChatMessage *)pvChatMessage;
        IChatClientWrapper *pClientWrapper = (IChatClientWrapper *)pvChatClientWrapper;

        // Continue routing message.
        HRESULT hr = pClientWrapper->SendMessage(__uuidof(IChatMessage), pMessage);
        if (FAILED(hr)) {
                pExtremeChat->ReportError(_T("Failed to route message to chat client."));
                return E_FAIL;
        }

        SAFERELEASE(pMessage);
        SAFERELEASE(pClientWrapper);
        return S_OK;
}

// IChatClientListener

STDMETHODIMP CExtremeChat::SendMessageResult(REFIID ChatClientWrapperIID, void *pClientWr
apper, long MessageID, BOOL Success) {
        WithScopeLocked(m_CritSec);
```

```
        if (!m_Initialized) {
                return E_UNEXPECTED;
        }

        LONG_MESSAGEINFO_MAP::iterator iter = m_MessageIDMap.find(MessageID);
        if (iter == m_MessageIDMap.end()) {
                ReportError(_T("Message ID not found for send message result."));
                return E_FAIL;
        }
        MessageInfo *pMessageInfo = iter->second;

        HRESULT hr = ContinueNewThread(SendMessageResultContinue, 4, this, pMessageInfo,
(void *)MessageID, (void *)Success);
        if (FAILED(hr)) {
                ReportError(_T("Failed to continue send message result in new thread."));
                return E_FAIL;
        }

        return S_OK;
}

HRESULT SendMessageResultContinue(CExtremeChat *pExtremeChat, void *pvMessageInfo, void *
pvMessageID, void *pvSuccess) {
        return pExtremeChat->SendMessageResult2((MessageInfo *)pvMessageInfo, (long)pvMes
sageID, (bool)pvSuccess);
}

HRESULT CExtremeChat::SendMessageResult2(MessageInfo *pMessageInfo, long MessageID, bool
Success) {
        // Notify delivery manager.
        BOOL Done = true;
        HRESULT hr = pMessageInfo->pDeliveryManagerWrapper->MessageRouted(__uuidof(IChatM
essage), pMessageInfo->pChatMessage, __uuidof(IChatLocation), pMessageInfo->pLocation, Su
ccess, &Done);
        if (FAILED(hr)) {
                ReportError(_T("Failed to notify delivery manager of message routing."));
                return E_FAIL;
        }

        WithScopeLocked(m_CritSec);

        // If delivery complete, notify original sender.
        if (Done) {
                m_pDeliveryManagerPool->DeallocateDeliveryManager(pMessageInfo->pDelivery
ManagerWrapper);

                MESSAGE_MESSAGINGCLIENTWRAP_MAP::iterator msgiter = m_MessageMap.find(pMe
ssageInfo->pChatMessage);
                if (msgiter == m_MessageMap.end()) {
                        ReportError(_T("Could not find successful message in map."));
                        return E_FAIL;
                }
                IMessagingClientWrapper *pMessagingClientWrapper = msgiter->second;

                hr = pMessageInfo->pChatMessage->SetStatus(Success ? CMS_SUCCEEDED : CMS_
FAILED);
                if (FAILED(hr)) {
                        ReportError(_T("Failed to set message status for routed message."
));
                        return E_FAIL;
                }

                hr = pMessagingClientWrapper->MessageNotification(__uuidof(IChatMessage),
 (IUnknown *)pMessageInfo->pChatMessage);
                if (FAILED(hr)) {
                        ReportError(_T("Failed to notify messaging client of routing comp
letion."));
                        return E_FAIL;
```

```
		}

		hr = MessageMapRemove(pMessageInfo->pChatMessage);
		if (FAILED(hr)) {
			ReportError(_T("Failed to remove routed message from map."));
			return E_FAIL;
		}
	}

	hr = MessageIDMapRemove(MessageID);
	if (FAILED(hr)) {
		ReportError(_T("Failed to remove message from routing table."));
		return E_FAIL;
	}

	return S_OK;
}

STDMETHODIMP CExtremeChat::UserStatusChanged(REFIID ChatClientIID, void *ChatClient, REFI
ID UserIID, void *pUser) {
	IChatUser *pChatUser = NULL;
	CoerceInterfacePointer(UserIID, pUser, IChatUser, pChatUser);

	WithScopeLocked(m_CritSec);

	if (!m_Initialized) {
		return E_UNEXPECTED;
	}

	for (MESSAGINGCLIENTWRAP_SET::iterator iter = m_MessagingClients.begin(); iter !=
 m_MessagingClients.end(); iter++) {
		IMessagingClientWrapper *pMessagingClientWrapper = *iter;

		// Continue in new thread.
		pMessagingClientWrapper->AddRef();
		pChatUser->AddRef();
		HRESULT hr = ContinueNewThread(UserStatusChangedContinue, 3, this, (void
*)pMessagingClientWrapper, (void *)pChatUser);
		if (FAILED(hr)) {
			ReportError(_T("Failed to continue reporting user status change i
n new thread."));
			return E_FAIL;
		}
	}

	SAFERELEASE(pChatUser);
	return S_OK;
}

HRESULT UserStatusChangedContinue(CExtremeChat *pExtremeChat, void *pvMessagingClientWrap
per, void *pvChatUser) {
	IMessagingClientWrapper *pMessagingClientWrapper = (IMessagingClientWrapper *)pvM
essagingClientWrapper;
	IChatUser *pChatUser = (IChatUser *)pvChatUser;

	// Continue user status changed.
	HRESULT hr = pMessagingClientWrapper->UserStatusChanged(__uuidof(IChatUser), stat
ic_cast<IUnknown *>(pChatUser));
	if (FAILED(hr)) {
		pExtremeChat->ReportError(_T("Failed to report user status change to mess
aging client."));
		return E_FAIL;
	}

	SAFERELEASE(pMessagingClientWrapper);
	SAFERELEASE(pChatUser);
	return S_OK;
}
```

ExtremeChat.cpp 16

```
STDMETHODIMP CExtremeChat::MessageReceived(REFIID ChatClientIID, void *ChatClient, REFIID
 MessageIID, void *pMessage) {
        IChatMessage *pChatMessage = NULL;
        CoerceInterfacePointer(MessageIID, pMessage, IChatMessage, pChatMessage);

        WithScopeLocked(m_CritSec);

        if (!m_Initialized) {
                return E_UNEXPECTED;
        }

        for (MESSAGINGCLIENTWRAP_SET::iterator iter = m_MessagingClients.begin(); iter !=
 m_MessagingClients.end(); iter++) {
                IMessagingClientWrapper *pMessagingClientWrapper = *iter;

                // Continue in new thread.
                pMessagingClientWrapper->AddRef();
                pChatMessage->AddRef();
                HRESULT hr = ContinueNewThread(MessageReceivedContinue, 3, this, (void *)
pMessagingClientWrapper, (void *)pChatMessage);
                if (FAILED(hr)) {
                        ReportError(_T("Failed to continue reporting message received in
new thread."));
                        return E_FAIL;
                }
        }

        SAFERELEASE(pChatMessage);
        return S_OK;
}

HRESULT MessageReceivedContinue(CExtremeChat *pExtremeChat, void *pvMessagingClientWrappe
r, void *pvChatMessage) {
        IMessagingClientWrapper *pMessagingClientWrapper = (IMessagingClientWrapper *)pvM
essagingClientWrapper;
        IChatMessage *pChatMessage = (IChatMessage *)pvChatMessage;

        // Continue message received notification.
        HRESULT hr = pMessagingClientWrapper->MessageReceived(__uuidof(IChatMessage), sta
tic_cast<IUnknown *>(pChatMessage));
        if (FAILED(hr)) {
                pExtremeChat->ReportError(_T("Failed to notify messaging client of incomi
ng message."));
                return E_FAIL;
        }

        SAFERELEASE(pMessagingClientWrapper);
        SAFERELEASE(pChatMessage);
        return S_OK;
}
```

IChatRegistry.idl 1

```
#ifndef ICHATREGISTRY_IDL
#define ICHATREGISTRY_IDL

#include "exchat.idl"

// PluginTypes: (ECPT = Extreme Chat Plugin Types)
//need to add search type
typedef short PluginTypes;
const PluginTypes ECPT_UI = 0x2; // Extreme Chat Type - UI Plugin
const PluginTypes ECPT_CHAT = 0x4; // Extreme Chat Type - Chat API Plugin
const PluginTypes ECPT_DELIVERY = 0x8; // Extreme Chat Type - Delivery Manager
const PluginTypes ECPT_MESSAGING = 0x10;// Extreme Chat Type - Non-UI Messaging
const PluginTypes ECPT_SEARCH = 0x20; // Extreme Chat Type - Search Plugin
const PluginTypes ECPT_OTHER = 0x1; // Extreme Chat Type - Miscellaneous Plugin
const PluginTypes ECPT_ALL = 0x3f;  //Extreme Chat Type - all types

// ChatRegistryInfoTypes: (ECRIT = Extreme Chat Registry Info Type)
typedef short ChatRegistryInfoTypes;
const ChatRegistryInfoTypes ECRIT_CLSID =     0x1;  // Plugin CLSID
const ChatRegistryInfoTypes ECRIT_TYPES =     0x2;  // Types of Plugin
const ChatRegistryInfoTypes ECRIT_NAME  =     0x4;  // Plugin Name
const ChatRegistryInfoTypes ECRIT_AUTHOR =    0x8;  // Author of Plugin
const ChatRegistryInfoTypes ECRIT_DESC =      0x10; // Plugin Description
const ChatRegistryInfoTypes ECRIT_DATE =      0x20; // Date of Plugin
const ChatRegistryInfoTypes ECRIT_VERSION =   0x40; // Version of Plugin
const ChatRegistryInfoTypes ECRIT_SELECTED = 0x80; // Plugin Selected?
const ChatRegistryInfoTypes ECRIT_ALL =       0xff; // All Info Types

struct ChatRegistryInfo {
  GUID clsid; // CLSID to start plugin
  PluginTypes types; // types of plugin
  LPOLESTR name; // human-readable name of plugin
  LPOLESTR author; // author of plugin
  LPOLESTR desc; // human-readable description
  LPOLESTR date; // date in what form?
  LPOLESTR versionNum; // version in what form?
  BOOL selected;  // is the plugin currently selected?
};


// Definition of the IChatRegistry Interface
// Interface for adding and removing system parts to and from the
// registry.

[
 object,
 uuid(980B105B-1AEB-11d3-B178-006094198F61),
 helpstring("IChatRegistry Interface"),
 pointer_default(unique)
]

interface IChatRegistry : IUnknown {
  [helpstring("method RegisterPlugin, either adds a new plugin or changes the information
 about an already-registered one")]
    HRESULT RegisterPlugin([in] BOOL NewObject, // is the plugin new or already
                                                // registered
                           [in] ChatRegistryInfoTypes fields,
                           [in] struct ChatRegistryInfo info);
  [helpstring("method UnregisterPlugin, removes an already registered plugin")]
    HRESULT UnregisterPlugin([in] GUID clsid);
  [helpstring("method EnumPlugins, returns all registered (not necessarily active, e.g. i
f user hasn't chosen them to currently run, that match one of input Plugin Types.  Info r
eturned is only info requested in ChatRequestInfoTypes argument.  If you ")]
    HRESULT EnumPlugins([in] PluginTypes types,
                        [in] ChatRegistryInfoTypes InfoRequested,
                        [out] short *pcNum,
                        [out, size_is(,*pcNum)] struct ChatRegistryInfo **Matches);
  [helpstring("method RegistryLookupPlugin, method to lookup any info for a single plugin
```

```
 by it's clsid which is the only unique ID field, call is failed with E_INVALIDARG if plu
gin doesn't exist in registry")]
        HRESULT RegistryLookupPlugin([in] GUID clsid,  // key to lookup plugin
                                     [in] ChatRegistryInfoTypes InfoRequested, // which info
will be meaningful in returned structure
                                                                    [out] struct ChatRegistr
yInfo *info); // pointer to return structure with answer
  [helpstring("method GetConfigurationString, retrieves the given configuration string fr
om the repository")]
        HRESULT GetConfigurationString([in] LPCOLESTR key, // "\" delimited list
                                                                    [out] LPOLESTR *value)
; // output, configuration string
  [helpstring("method SetConfigurationString, sets the given configuration string for lat
er retrieval")]
        HRESULT SetConfigurationString([in] LPCOLESTR key, // "\" delimited list
                                                                    [in] LPCOLESTR value);
 // the configuration string
  [helpstring("method Save, indicates a desire to save state, successful return does not
mean that state was saved")]
    HRESULT Save();
};

[
 object,
 uuid(980B105C-1AEB-11d3-B178-006094198F61),
 dual,
 helpstring("IDispChatRegistry Interface"),
 pointer_default(unique)
]
interface IDispChatRegistry : IDispatch
{
  [id(1), helpstring("method GetConfigurationString, retrieves the given configuration st
ring from the repository")]
        HRESULT GetConfigString([in] BSTR key, // "\" delimited list
                                                    [out, retval] BSTR *value); // output
, configuration string
  [id(2), helpstring("method SetConfigurationString, sets the given configuration string
for later retrieval")]
        HRESULT SetConfigString([in] BSTR key, // "\" delimited list
                                                    [in] BSTR value); // the configuratio
n string
};


#endif
```

IChatRegistry.idl 1

```
#ifndef ICHATREGISTRY_IDL
#define ICHATREGISTRY_IDL

#include "exchat.idl"

// PluginTypes: (ECPT = Extreme Chat Plugin Types)
//need to add search type
typedef short PluginTypes;
const PluginTypes ECPT_UI = 0x2; // Extreme Chat Type - UI Plugin
const PluginTypes ECPT_CHAT = 0x4; // Extreme Chat Type - Chat API Plugin
const PluginTypes ECPT_DELIVERY = 0x8; // Extreme Chat Type - Delivery Manager
const PluginTypes ECPT_MESSAGING = 0x10;// Extreme Chat Type - Non-UI Messaging
const PluginTypes ECPT_SEARCH = 0x20; // Extreme Chat Type - Search Plugin
const PluginTypes ECPT_OTHER = 0x1; // Extreme Chat Type - Miscellaneous Plugin
const PluginTypes ECPT_ALL = 0x3f;  //Extreme Chat Type - all types

// ChatRegistryInfoTypes: (ECRIT = Extreme Chat Registry Info Type)
typedef short ChatRegistryInfoTypes;
const ChatRegistryInfoTypes ECRIT_CLSID =     0x1;  // Plugin CLSID
const ChatRegistryInfoTypes ECRIT_TYPES =     0x2;  // Types of Plugin
const ChatRegistryInfoTypes ECRIT_NAME  =     0x4;  // Plugin Name
const ChatRegistryInfoTypes ECRIT_AUTHOR =    0x8;  // Author of Plugin
const ChatRegistryInfoTypes ECRIT_DESC =      0x10; // Plugin Description
const ChatRegistryInfoTypes ECRIT_DATE =      0x20; // Date of Plugin
const ChatRegistryInfoTypes ECRIT_VERSION =   0x40; // Version of Plugin
const ChatRegistryInfoTypes ECRIT_SELECTED = 0x80; // Plugin Selected?
const ChatRegistryInfoTypes ECRIT_ALL =       0xff; // All Info Types

struct ChatRegistryInfo {
  GUID clsid; // CLSID to start plugin
  PluginTypes types; // types of plugin
  LPOLESTR name; // human-readable name of plugin
  LPOLESTR author; // author of plugin
  LPOLESTR desc; // human-readable description
  LPOLESTR date; // date in what form?
  LPOLESTR versionNum; // version in what form?
  BOOL selected;  // is the plugin currently selected?
};


// Definition of the IChatRegistry Interface
// Interface for adding and removing system parts to and from the
// registry.

[
 object,
 uuid(980B105B-1AEB-11d3-B178-006094198F61),
 helpstring("IChatRegistry Interface"),
 pointer_default(unique)
]

interface IChatRegistry : IUnknown {
  [helpstring("method RegisterPlugin, either adds a new plugin or changes the information
 about an already-registered one")]
    HRESULT RegisterPlugin([in] BOOL NewObject, // is the plugin new or already
                                                // registered
                           [in] ChatRegistryInfoTypes fields,
                           [in] struct ChatRegistryInfo info);
  [helpstring("method UnregisterPlugin, removes an already registered plugin")]
    HRESULT UnregisterPlugin([in] GUID clsid);
  [helpstring("method EnumPlugins, returns all registered (not necessarily active, e.g. i
f user hasn't chosen them to currently run, that match one of input Plugin Types.  Info r
eturned is only info requested in ChatRequestInfoTypes argument.  If you ")]
    HRESULT EnumPlugins([in] PluginTypes types,
                        [in] ChatRegistryInfoTypes InfoRequested,
                        [out] short *pcNum,
                        [out, size_is(,*pcNum)] struct ChatRegistryInfo **Matches);
  [helpstring("method RegistryLookupPlugin, method to lookup any info for a single plugin
```

```
 by it's clsid which is the only unique ID field, call is failed with E_INVALIDARG if plu
gin doesn't exist in registry")]
        HRESULT RegistryLookupPlugin([in] GUID clsid,  // key to lookup plugin
                                     [in] ChatRegistryInfoTypes InfoRequested, // which info
will be meaningful in returned structure
                                                                              [out] struct ChatRegistr
yInfo *info); // pointer to return structure with answer
  [helpstring("method GetConfigurationString, retrieves the given configuration string fr
om the repository")]
        HRESULT GetConfigurationString([in] LPCOLESTR key, // "\" delimited list
                                                                               [out] LPOLESTR *value)
; // output, configuration string
  [helpstring("method SetConfigurationString, sets the given configuration string for lat
er retrieval")]
        HRESULT SetConfigurationString([in] LPCOLESTR key, // "\" delimited list
                                                                               [in] LPCOLESTR value);
 // the configuration string
  [helpstring("method Save, indicates a desire to save state, successful return does not
mean that state was saved")]
    HRESULT Save();
};

[
 object,
 uuid(980B105C-1AEB-11d3-B178-006094198F61),
 dual,
 helpstring("IDispChatRegistry Interface"),
 pointer_default(unique)
]
interface IDispChatRegistry : IDispatch
{
  [id(1), helpstring("method GetConfigurationString, retrieves the given configuration st
ring from the repository")]
        HRESULT GetConfigString([in] BSTR key, // "\" delimited list
                                                [out, retval] BSTR *value); // output
, configuration string
  [id(2), helpstring("method SetConfigurationString, sets the given configuration string
for later retrieval")]
        HRESULT SetConfigString([in] BSTR key, // "\" delimited list
                                                [in] BSTR value); // the configuratio
n string
};


#endif
```

IDeliveryManager.idl 1

```
#ifndef _IDELIVERYMANAGER_IDL
#define _IDELIVERYMANAGER_IDL

#include "exchat.idl"

/*

IDeliveryManager  <-->  IDeliveryManagerWrapper  <-->  IDeliveryManagerListener

Delivery managers are used as follows:

Parent                          Delivery Manager          Note
------                          ----------------          ----

[SetActiveChatLocations] -->                                Must be called before first res
et.

Reset                           -->                       Reset internal state.

StartDelivery                   -->                       Start delivery for a message.

while(!Done) {

                                <--     (RouteMessage)*   Send the message using specified
                                                                                    locat
ion.

(MessageRouted)*                -->                       RouteMessage result, ask if done.

}

*/

// IDeliveryManager

[
        object,
        uuid(94C38BCC-388F-11D3-BC33-0004AC77F55A),
        helpstring("IDeliveryManager Interface"),
        pointer_default(unique)
]
interface IDeliveryManager : IPlugin {
        HRESULT SetActiveChatLocations([in] REFIID EnumLocationsIID, [in, iid_is(EnumLoca
tionsIID)] void *pEnumLocations);
        HRESULT Reset();
        HRESULT StartDelivery([in] REFIID MessageIID, [in, iid_is(MessageIID)] void *pMes
sage);
        HRESULT MessageRouted([in] REFIID MessageIID, [in, iid_is(MessageIID)] void *pMes
sage, [in] REFIID LocationIID, [in, iid_is(LocationIID)] void *pLocation, [in] BOOL Succe
eded, [out] BOOL *pDone);
};

// IDeliveryManagerWrapper

[
        object,
        uuid(CADAB760-4062-11d3-89B7-0020355E62E2),
        helpstring("IDeliveryManagerWrapper Interface"),
        pointer_default(unique)
]
interface IDeliveryManagerWrapper : IPluginWrapper {
        // Called by plugin.
        HRESULT RouteMessage([in] REFIID MessageIID, [in, iid_is(MessageIID)] void *pMess
age, [in] REFIID LocationIID, [in, iid_is(LocationIID)] void *pLocation);
        // Called by parent.
        HRESULT SetActiveChatLocations([in] REFIID EnumLocationsIID, [in, iid_is(EnumLoca
tionsIID)] void *pEnumLocations);
        HRESULT Reset();
```

IDeliveryManager.idl

2

```
        HRESULT StartDelivery([in] REFIID MessageIID, [in, iid_is(MessageIID)] void *pMes
sage);
        HRESULT MessageRouted([in] REFIID MessageIID, [in, iid_is(MessageIID)] void *pMes
sage, [in] REFIID LocationIID, [in, iid_is(LocationIID)] void *pLocation, [in] BOOL Succe
eded, [out] BOOL *pDone);
};

// IDeliveryManagerListener

[
        object,
        uuid(94C38BF4-388F-11D3-BC33-0004AC77F55A),

        helpstring("IDeliveryManagerListener Interface"),
        pointer_default(unique)
]
interface IDeliveryManagerListener : IUnknown {
        HRESULT RouteMessage([in] REFIID DeliveryManagerWrapperIID, [in, iid_is(DeliveryM
anagerWrapperIID)] void *pDeliveryWrapperManager, [in] REFIID MessageIID, [in, iid_is(Mes
sageIID)] void *pMessage, [in] REFIID LocationIID, [in, iid_is(LocationIID)] void *pLocat
ion);
};

#endif // _IDELIVERYMANAGER_IDL
```

IDeliveryManagerWrapperImpl.h 1

```
#ifndef _IDELIVERYMANAGERWRAPPERIMPL_H
#define _IDELIVERYMANAGERWRAPPERIMPL_H

#include "com_macros.h"
#include "exchat.h"
#include "exchatimpl.h"

class ATL_NO_VTABLE IDeliveryManagerWrapperImpl :
        public IPluginWrapperImpl,
        public IDeliveryManagerWrapper
{

public:

        HRESULT FinalConstruct() {
                IPluginWrapperImpl::FinalConstruct();
                m_pWrapper = static_cast<IDeliveryManagerWrapper *>(this);
                m_pWrapper->AddRef();
                m_pListener = NULL;
                m_pDeliveryManager = NULL;
                m_pEnumChatLocation = NULL;
                return S_OK;
        }

        HRESULT FinalRelease() {
                IPluginWrapperImpl::FinalRelease();
                SAFERELEASE(m_pListener);
                SAFERELEASE(m_pWrapper);
                SAFERELEASE(m_pDeliveryManager);
                SAFERELEASE(m_pEnumChatLocation);
                return S_OK;
        }

        // IPluginWrapper

        STDMETHOD(SetParent)(REFIID ParentIID, void *pParent) {
                HRESULT hr = IPluginWrapperImpl::SetParent(ParentIID, pParent);
                if (FAILED(hr)) {
                        return hr;
                }
                if (!GUIDsAreEqual(ParentIID, __uuidof(IDeliveryManagerListener))) {
                        return E_INVALIDARG;
                }
                SAFERELEASE(m_pListener);
                m_pListener = static_cast<IDeliveryManagerListener *>(pParent);
                m_pListener->AddRef();
                return S_OK;
        }

        STDMETHOD(SetPlugin)(REFIID PluginIID, void *pPlugin) {
                HRESULT hr = IPluginWrapperImpl::SetPlugin(PluginIID, pPlugin);
                if (FAILED(hr)) {
                        return hr;
                }
                if (!GUIDsAreEqual(PluginIID, __uuidof(IDeliveryManager))) {
                        return E_INVALIDARG;
                }
                SAFERELEASE(m_pDeliveryManager);
                m_pDeliveryManager = static_cast<IDeliveryManager *>(pPlugin);
                m_pDeliveryManager->AddRef();
                hr = m_pDeliveryManager->SetParent(__uuidof(IDeliveryManagerWrapper), sta
tic_cast<IDeliveryManagerWrapper *>(this));
                if (FAILED(hr)) {
                        return E_FAIL;
                }
                return S_OK;
        }
```

IDeliveryManagerWrapperImpl.h 2

```
        // IDeliveryManagerWrapper

        STDMETHOD(RouteMessage)(REFIID MessageIID, void *pMessage, REFIID LocationIID, vo
id *pLocation) {
                if ((m_pListener == NULL) || (m_pDeliveryManager == NULL)) {
                        return E_FAIL;
                }
                return m_pListener->RouteMessage(__uuidof(IDeliveryManagerWrapper), m_pWr
apper, MessageIID, pMessage, LocationIID, pLocation);
        }

        // Save the new value, don't apply until the next Reset.
        STDMETHOD(SetActiveChatLocations)(REFIID EnumLocationsIID, void *pEnumLocations)
{
                if (!GUIDsAreEqual(EnumLocationsIID, __uuidof(IEnumChatLocation))) {
                        return E_INVALIDARG;
                }
                // Release any existing value, then update.
                SAFERELEASE(m_pEnumChatLocation);
                m_pEnumChatLocation = static_cast<IEnumChatLocation *>(pEnumLocations);
                m_pEnumChatLocation->AddRef();
                return S_OK;
        }

        STDMETHOD(Reset)() {
                if (m_pDeliveryManager == NULL) {
                        return E_UNEXPECTED;
                }
                // Check to see if active plugins should be updated.
                if (m_pEnumChatLocation != NULL) {
                        HRESULT hr = m_pDeliveryManager->SetActiveChatLocations(__uuidof(
IEnumChatLocation), (void *)m_pEnumChatLocation);
                        if (FAILED(hr)) {
                                return E_FAIL;
                        }
                        SAFERELEASE(m_pEnumChatLocation);
                }
                return m_pDeliveryManager->Reset();
        }

        STDMETHOD(StartDelivery)(REFIID MessageIID, void *pMessage) {
                if (m_pDeliveryManager == NULL) {
                        return E_UNEXPECTED;
                }
                return m_pDeliveryManager->StartDelivery(MessageIID, pMessage);
        }

        STDMETHOD(MessageRouted)(REFIID MessageIID, void *pMessage, REFIID LocationIID, v
oid *pLocation, BOOL Succeeded, BOOL *pDone) {
                if (m_pDeliveryManager == NULL) {
                        return E_FAIL;
                }
                return m_pDeliveryManager->MessageRouted(MessageIID, pMessage, LocationII
D, pLocation, Succeeded, pDone);
        }

protected:

        IDeliveryManagerListener *m_pListener;
        IDeliveryManagerWrapper *m_pWrapper;
        IDeliveryManager *m_pDeliveryManager;
        IEnumChatLocation *m_pEnumChatLocation;

};

#endif // _IDELIVERYMANAGERWRAPPERIMPL_H
```